Finite Model Finding for Parameterized Verification

Alexei Lisitsa

Department of Computer Science
University of Liverpool, UK
A.Lisitsa@liverpool.ac.uk

Abstract. In this paper we investigate to what extent a very simple and natural "reachability as deducibility" approach, originating in research on formal methods for security, is applicable to the automated verification of large classes of infinite state and parameterized systems. This approach is based on modeling the reachability between (parameterized) states as deducibility between suitable encodings of states by formulas of first-order predicate logic. The verification of a safety property is reduced to the purely logical problem of finding a countermodel for a first-order formula. This task is then delegated then to generic automated finite model building procedures. In this paper we first establish the relative completeness of the finite countermodel finding method (FCM) for a class of parameterized linear arrays of finite automata. The method is shown to be at least as powerful as known methods based on monotonic abstraction and symbolic backward reachability. Further, we extend the relative completeness of the approach and show that it can solve all safety verification problems which can be solved by regular model checking.

1 Introduction

The verification of infinite state systems and parameterized systems is, in general, an undecidable algorithmic problem. That means the search for efficient procedures to tackle the larger and larger subclasses of verification tasks will never end. In this paper we investigate to what extent a very simple and natural "reachability as deducibility" approach is applicable to the verification such systems. Consider an encoding $e : s \mapsto \varphi_s$ of states of a transition system $\mathcal{S} = \langle S, \rightarrow \rangle$ by formulae of first-order predicate logic satisfying the following property. The state s' is reachable from s, i.e. $s \rightarrow^* s'$ if and only if $\varphi_{s'}$ is the logical consequence of φ_s, that is $\varphi_s \models \varphi_{s'}$ or $\varphi_s \vdash \varphi_{s'}$. Under such assumptions establishing reachability amounts to theorem proving, while deciding non-reachability, becomes theorem disproving. To verify a safety property, i.e non-reachability of *unsafe* states, it is sufficient to *disprove* a formula of the form $\phi \rightarrow \psi$. Also, in the case of safety verification already half of the assumption on the encoding is suffcient: $(s \rightarrow^* s') \Rightarrow (\varphi_s \vdash \varphi_{s'})$. The task of disproving can be delegated then to generic model finding procedures for first-order predicate logic [5].

Such an approach to verification originated within research on formal methods for the analysis of cryptographic protocols [23,22,11,15]. Being unaware of these developments in the verification of cryptographic protocols and coming from a different perspective we re-invented the finite countermodel finding approach and applied it in a different context of verification of parameterized and infinite state systems [17,18,19].

We show in [19] that the parallel composition of a complete finite model finder and a complete theorem prover provides a decision procedure for safety properties of lossy channel systems [3] under appropriate encoding. Using a finite model finder, Mace4, [20] coupled with a theorem prover Prover9 [20] we successfully applied the method to the verification of alternating bit protocol, specified within a lossy channel system; all parameterized cache coherence protocols from [7]; series of coverability and reachability tasks conserning Petri Nets; parameterized Dining Philosophers Problem (DPP) and to parameterized linear systems (arrays) of finite automata.

Despite the wide range of parameterized verification tasks being tackled successfully by the method, the only result concerning completeness presented so far is that on the verification of lossy channel systems [19]. The aim of this paper is to investigate further the completeness of the finite countermodel finding method for much larger classes of parameterized verification tasks. Note that we focus here on *relative* completeness with respect to well-known methods. To introduce the method we present as case study in Section 3 the details of automated verification of a parameterized mutual exclusion protocol, which is an instance of parameterized model defined in Section 2. Further, we present an appropriate translation of verification tasks for the parameterized systems of finite automata arranged in linear arrays into formulae of first-order predicate logic (subsection 4.1). We show, in subsection 4.2, that the proposed finite countermodel finding method is at least as powerful as the methods based on monotone abstraction and symbolic backward reachability analysis [1] for this class of verification problems. Further, in Section 5 we extend the relative completeness of the approach and show that it can solve all safety verification problems which can be solved by a traditional *regular model checking* [21]. In Section 6 we discuss related work and Section 7 concludes the paper.

1.1 Preliminaries

We assume that the reader is familiar with the the basics of first-order logic and algebra. In particular, we use without definitions the following concepts: first-order predicate logic, first-order models, interpretations of relational, functional and constant symbols, satisfaction $M \models \varphi$ of a formula φ in a model M, semantical consequence $\varphi \models \psi$, deducibility (derivability) $\vdash$ in first-order logic, monoid, homomorphism, finite automata and the algebraic characterization of regular languages. We denote interpretations by square brackets, so, for example, $[f]$ denotes an interpretation of a functional symbol f in a model. We also use the existence of *complete* finite model finding procedures for the first-order predicate logic [5,20], which given a first-order sentence φ eventually produce a finite model for φ if such a model exists.

2 Parameterized linear arrays of automata

The computational model we first consider in this paper consists of parameterized systems of finite automata arranged in linear arrays [1]. Formally, a parameterized system $\mathcal{P}$ is a pair (Q, T), where Q is a finite set of local states of processes and T is finite set of transition rules. Every transition rule has one of the following forms

- $q \rightarrow q'$ where $q, q' \in Q$;
- $\mathcal{G} : q \rightarrow q'$, where $q, q' \in Q$ and $\mathcal{G}$ is a condition of the form $\forall_I J$, or $\exists_I J$

Here $J \subseteq Q$ and I is an indicator of the context, and it may be one of the following: L (for Left), R (for Right), or LR (for both Left and Right).

Given a parameterized system $\mathcal{P} = (Q, T)$ the *configuration* of the system is a word $\bar{c} = c_1 c_2 \ldots c_n \in Q^*$. Intuitively, the configuration represents the local states of a family of n finite state automata (processes) arranged in a linear array, so, for example $c_i \in Q$ is a local state of automaton at position i in the array.

For a configuration $\bar{c} = c_1 \ldots c_n$, position $i : 1 \leq i \leq n$ and a condition, we define $\models$, a *satisfaction* relation:

- $(\bar{c}, i) \models \forall_L J$ iff $\forall k < i \;\; c_k \in J$;
- $(\bar{c}, i) \models \forall_R J$ iff $\forall k > i \;\; c_k \in J$;
- $(\bar{c}, i) \models \forall_{LR} J$ iff $(\bar{c}, i) \models \forall_L J$ and $(\bar{c}, i) \models \forall_R J$
- $(\bar{c}, i) \models \exists_L J$ iff $\exists k < i \;\; c_k \in J$;
- $(\bar{c}, i) \models \exists_R J$ iff $\exists k > i \;\; c_k \in J$;
- $(\bar{c}, i) \models \exists_{LR} J$ iff $(\bar{c}, i) \models \exists_L J$ or $(\bar{c}, i) \models \exists_R J$

A parameterized system $\mathcal{P} = (Q, T)$ induces a transition relation $\rightarrow_{\mathcal{P}}$ on the set C of all configurations as follows. For two configurations $\bar{c} \rightarrow_{\mathcal{P}} \bar{c}'$ holds iff either

- $q \rightarrow q'$ is a transition in T and for some $i : 1 \leq i \leq n \; c_i = q, c_i' = q'$ and $\forall j \neq i \;\; c_j = c_j'$, or
- $\mathcal{G} : q \rightarrow q'$ is a transition in T and for some $i : 1 \leq i \leq n \; c_i = q, c_i' = q'$, $(\bar{c}, i) \models \mathcal{G}$ and $\forall j \neq i \;\; c_j = c_j'$

The general form of the verification problem we consider here is as follows.

> *Given: A parameterized system $\mathcal{P} = (Q, T)$, a set $In \subseteq C$ of initial configurations, a set $B \subseteq C$ of* bad *configurations.*
> *Question: Are there any configurations $c \in In$ and $c' \in B$ such that c' is reachable from c in $\mathcal{P}$, i.e. for which $c \rightarrow_{\mathcal{P}}^* c'$ holds?*

A negative answer for the above question means the safety property ("not B") holds for the parameterized system.

3 Case study

3.1 Mutual Exclusion Protocol

We consider the verification of the parameterized mutual exclusion protocol which was used as an illustrative example in [1]. This protocol is specified as a parameterized system $\mathcal{ME} = (Q, T)$, where $Q = \{green, black, blue, red\}$ and T consists of the following transitions:

- $\forall_{LR}\{green, black\} : green \rightarrow black$
- $black \rightarrow blue$
- $\exists_L\{black, blue, red\} : blue \rightarrow blue$
- $\forall_L\{green\} : blue \rightarrow red$
- $red \rightarrow black$
- $black \rightarrow green$

The set of initial configurations $In = green^*$ consists of all configurations with all automata in $green$ states. The safety property we would like to check is a mutual exclusion of red states, i.e. in any reachable configuration, there are no more than one automaton in the red state. The set B of bad configurations is defined then by straightforward regular expression $B = Q^* \, red \, Q^* \, red \, Q^*$.

3.2 First-Order encoding

We define a translation of the above parameterized system into a set of formulae $\Phi_{\mathcal{P}}$ of first-order logic. The vocabulary of $\Phi_{\mathcal{P}}$ consists of

- constants $green$, $blue$, $black$, red and e
- one binary functional symbol $*$
- unary predicates R, G, GB

Given a configuration $\bar{c} = c_1 \ldots c_n$ of $\mathcal{P}$ define its term translation as $t_{\bar{c}} = c_1 * \ldots * c_n$. It is well-defined modulo the associativity of $'*'$ which we will specify in the formula, and uses an assumption that in the language we have all the elements of Q as constants.

The intended meaning of atomic formula $R(t_{\bar{c}})$ is that the configuration $\bar{c}$ is reachable, while $G(t_{\bar{c}})$ and $GB(t_{\bar{c}})$ mean $\bar{c}$ has only automata in $green$ states, and $\bar{c}$ has only automata in $green$ or $black$ states, respectively.

Let $\Phi_{\mathcal{P}}$ be a set of the following formulae, which are all assumed to be universally closed:

- $(x * y) * z = x * (y * z)$
- $e * x = x * e = x$

  ($*$ is a monoid operation and e is a unit of a monoid)

- $G(e)$
- $G(x) \rightarrow G(x * green)$

  (specification of configurations with all $green$ states)

- $GB(e)$
- $GB(x) \rightarrow GB(x * green)$
- $GB(x) \rightarrow GB(x * black)$

  (specification of configurations with all states being $green$ or $black$)

- $G(x) \rightarrow R(x)$

(initial state assumption: "allgreen" configurations are reachable)

- $(R((x * green) * y) \,\&\, GB(x) \,\&\, GB(y)) \rightarrow R((x * black) * y)$
- $R((x * black) * y) \rightarrow R((x * blue) * y)$
- $R((x * blue) * y) \,\&\, (x = (z * black) * w) \rightarrow R((x * blue) * y)$
- $R((x * blue) * y) \,\&\, (x = (z * blue) * w) \rightarrow R((x * blue) * y)$
- $R((x * blue) * y) \,\&\, (x = (z * red) * w) \rightarrow R((x * blue) * y)$
- $R((x * blue) * y) \,\&\, G(x) \rightarrow R((x * red) * y)$
- $R((x * red) * y) \rightarrow R((x * black) * y)$
- $R((x * black) * y) \rightarrow R((x * green) * y)$

(specification of reachability by one step transitions from T; one formula per transition, except the case with an existential condition, where three formulae are used)

Now we have a key proposition

Proposition 1 (adequacy of encoding). *If a configuration $\bar{c}$ is reachable in $\mathcal{ME}$ then $\Phi_{\mathcal{P}} \vdash R(t_{\bar{c}})$*

Proof By straightforward induction on the length of transition sequences in $\mathcal{ME}$ $\square$

3.3 Verification

It follows now, that to establish safety property of the protocol (mutual exclusion), it does suffice to show that $\Phi_{\mathcal{P}} \not\vdash \exists x \exists y \exists z R((((x * red) * y) * red) * z)$. Indeed, if, on the contrary, some bad configuration $\bar{c}$ would be reachable, then by Proposition 1 we would have for some terms t_1, t_2, t_3 that $\Phi_{\mathcal{P}} \vdash R(t_{\bar{c}})$ where $t_{\bar{c}} = (((t_1 * red) * t_2) * red) * t_3$, and therefore $\Phi_{\mathcal{P}} \vdash \exists x \exists y \exists z R((((x * red) * y) * red) * z)$. Further, to show non-deducibility, it is sufficient to find a countermodel for $\Phi_{\mathcal{P}} \rightarrow \exists x \exists y \exists z R((((x * red) * y) * red) * z)$.

Now we propose to delegate this last task to an automated procedure for finite model finding, which would search for a *finite* model for

$$\Phi_{\mathcal{P}} \wedge \neg \exists x \exists y \exists z R((((x * red) * y) * red) * z)$$

In the practical implementation of this scheme we used a finite model finder Mace4 [20], which was able to find a required model in 0.03 seconds. Actual input for Mace4 and further details can be found in [18].

A priori, to disprove some implication in first-order logic, searching for finite countermodels may be not sufficient, for such countermodels may inevitably be infinite. It has turned out empirically though that for many known parameterized (classes of) problems, finite model finding is, indeed, both sufficient and efficient. In [17] we established the first result on completeness of the method for a particular class of infinite state verification tasks. Here we demonstrate further results on *relative* completeness.

4 Correctness and Completeness

4.1 First-Order Encoding for General Case

In the general form of the verification problem above we have to agree what are the allowed sets of initial and bad configurations can be, and what are their constructive representations. Here we assume that

– one of the local states $q_0 \in Q$ is singled out as an initial state, and the set $Init$ of initial configurations is always q_0^*, i.e. it consists of all configurations that have all the automata in their local initial states;
– The set B of bad configurations is defined by a finite set of words $F \subseteq Q^*$: $B = \{\bar{c} \mid \exists \bar{w} \in F \ \wedge \ \bar{w} \preceq \bar{c}\}$, where $\bar{w} \preceq \bar{w}'$ denotes that $\bar{w}$ is a (not necessarily contiguous) subword of $\bar{w}'$. The elements of such F are called *generators* of B.

To illustrate this last point, in our Case Study above, the set of bad configurations B is defined by an F consisting of one word with two symbols $red\ red$.

Given a parameterized system $\mathcal{P} = (\mathcal{Q}, \mathcal{T})$, an intial local state $q_0 \in Q$, a finite set of words F, we translate all of this into a set of formulae in first-order logic.

The vocabulary consists of

– constants for all elements of Q plus one distinct constant, so we take $Q \cup \{e\}$, with $e \notin Q$ as the set of constants;
– the binary functional symbol $*$;
– the unary relational symbol In;
– the unary relational symbol R;
– for every condition $\forall_I J$ in the transitions from T a unary relational symbol P^J

Let Φ_P be the set of the following formulae, which are all assumed to be universally closed:

– $(x * y) * z = x * (y * z)$
– $e * x = x * e = x$

– $In(e)$
– $In(x) \rightarrow In(x * q_0)$
– $In(x) \rightarrow R(x)$

For every condition $\forall_I J$ in the transitions from T:

– $P^J(e)$
– $wedge_{q \in J}(P^J(x) \rightarrow P^J(x * q))$

For every unconditional transition $q_1 \rightarrow q_2$ from T:

– $R((x * q_1) * y) \rightarrow R((x * q_2) * y)$

For every conditional transition $\forall_L J\ (q_1 \rightarrow q_2)$ from T:

– $(R((x * q_1) * y) \wedge P^J(x)) \rightarrow R((x * q_2) * y)$

For every conditional transition $\forall_R J\ (q_1 \rightarrow q_2)$ from T:

– $(R((x * q_1) * y) \wedge P^J(y)) \rightarrow R((x * q_2) * y)$

For every conditional transition $\forall_{LR} J \ (q_1 \rightarrow q_2)$ from T:

– $(R((x * q_1) * y) \wedge P^J(x) \wedge P^J(y)) \rightarrow R((x * q_2) * y)$

For every conditional transition $\exists_L J \ (q_1 \rightarrow q_2)$ from T:

– $\wedge_{q \in J}(R(x * q_1) * y) \wedge (x = (z * q) * w)) \rightarrow R((x * q_2) * y)$

For every conditional transition $\exists_R J \ (q_1 \rightarrow q_2)$ from T:

– $\wedge_{q \in J}(R(x * q_1) * y) \wedge (y = (z * q) * w)) \rightarrow R((x * q_2) * y)$

For every conditional transition $\exists_{LR} J \ (q_1 \rightarrow q_2)$ from T:

– $\wedge_{q \in J}(R(x * q_1) * y) \wedge ((x = (z * q) * w) \vee (y = (z * q) * w))) \rightarrow R((x * q_2) * y)$

That concludes the definition of $\Phi_{\mathcal{P}}$. Next, for a word $\bar{w} = w_1, \ldots, w_n \in Q^*$ we define (up to the associativity of $*$) the formula $\psi_{\bar{w}}$ as $R(x_0 * w_1 * x_1 * \ldots * x_{n-1} * w_n * x_n)$ where $x_0, \ldots x_n$ are variables. Finally, we define Ψ_F as $\exists \bar{x} \vee_{\bar{w} \in F} \psi_{\bar{w}}$ (here we assume that all variables are bound by existential quantifiers).

The following generalization of Proposition 1 holds.

Proposition 2 (adequacy of encoding). *If configuration $\bar{c}$ is reachable in $\mathcal{P}$ then $\Phi_{\mathcal{P}} \vdash R(t_{\bar{c}})$*

Proof By straightforward induction on the length of the transition sequences $\square$.

Corollary 1 (correctness of the method). *If $\Phi_{\mathcal{P}} \not\vdash \Psi_F$ then the answer to the question of the verification problem is negative, that is no bad configuration is reachable from any of the initial configurations, and therefore, the safety property holds.*

4.2 Relative completeness

Here we show that on the the class of the verification problems described above our proposed method is at least as powerful as the standard approach based on monotone abstraction [1]. Specifically, if for a parameterized system $\mathcal{P}$ the approach [1] proves a safety property, then our method based on finite countermodel finding will also succeed in establishing this property, provided a *complete* finite model finding procedure is used.

First, we briefly outline the monotone abstraction approach. Given a parameterized system $\mathcal{P} = (Q, T)$ and corresponding transition relation $\rightarrow_{\mathcal{P}}$ on the configurations withing $\mathcal{P}$, [1] defines the monotonic abstraction $\rightarrow^A_{\mathcal{P}}$ of $\rightarrow_{\mathcal{P}}$ as follows.

We have $\bar{c}_1 \rightarrow^A_{\mathcal{P}} \bar{c}_2$ iff there exists a configuration $\bar{c}'_1$ such that $\bar{c}'_1 \preceq \bar{c}_1$ and $\bar{c}' \rightarrow_{\mathcal{P}} c_2$.

Such defined $\to_{\mathcal{P}}^A$ is an over-approximation of $\to_{\mathcal{P}}$. To establish the safety property, i.e to get a negative answer to the question of the verification problem above, [1] proposes using a symbolic backward reachability algorithm for monotone abstraction. Starting with an upwards closed (wrt to $\preceq$) set of bad configurations $B = \{\bar{c} \mid \exists \bar{w} \in F \ \wedge \ \bar{w} \preceq \bar{c}\}$, the algorithm proceeds iteratively with the computation of the sets of configurations backwards reachable along $\to_{\mathcal{P}}^A$ from B:

- $U_0 = B$
- $U_{i+1} = U_i \cup Pre(U_i)$

where $Pre(U) = \{\bar{c} \mid \exists \bar{c}' \in U \ \wedge \ \bar{c} \to_{\mathcal{P}}^A \bar{c}'\}$. Since the relation $\preceq$ is a *well quasi-ordering* [1] this iterative process is guaranteed to stabilize, i.e $U_{n+1} = U_n$ for some finite n. During the computation each U_i is represented symbolically by a finite sets of generators. Once the process stabilized on some U the check is performed on whether $Init \cap U = \emptyset$. If this condition is satisfied then the safety is established, for no bad configuration can be reached from intial configurations via $\to_{\mathcal{P}}^A$ and, a fortiori, via $\to_{\mathcal{P}}$.

Theorem 1 (relative completeness). *Given a parameterized system $\mathcal{P} = (Q, T)$ and the set of bad configurations $B = \{\bar{c} \mid \exists \bar{w} \in F \ \wedge \ \bar{w} \preceq \bar{c}\}$. Assume the algorithm described above terminates with $Init \cap U = \emptyset$. Then there exists a* finite model *for $\Phi_{\mathcal{P}} \wedge \neg \Psi_F$*

Proof. First we observe that since $U \subseteq Q^*$ has a finite set of generators, it is a *regular* set. According to the algebraic characterization of regular sets, there exists a *finite* monoid $\mathcal{M} = (M, \circ)$, a subset $S \subseteq M$ and a homomorphism $h : Q^* \to \mathcal{M}$ from the free monoid Q^* to $\mathcal{M}$ such that $U = \{\bar{w} \mid \bar{w} \in Q^* \wedge h(\bar{w}) \in S\}$. We set M to be domain of the required finite model.

Now we define interpretations of constants: for $q \in Q$ $[q] = h(q)$ and $[e] = \underline{1}$, where $\underline{1}$ is an unit element of the monoid.

The interpretation $[*]$ of $*$ is a monoid operation $\circ$. We define an interpretation of R as $[R] = M - S$.

We define an interpretation of In inductively: $[In]$ is the least subset of M satisfying $\underline{1} \in [In]$ and $\forall x \in [In] \ x \circ [q_0] \in [In]$.

An interpretation of P^J is defined inductively as follows. $[P^J]$ is a least subset of M satisfying $\underline{1} \in [P^j]$ and $\forall x \in [P^J] \ \forall q \in J \ x \circ [q] \in [P^J]$. That concludes the definition of the finite model, which we denote by $\mathfrak{M}$. The key property of the model is given by the following lemma.

Lemma 1. $h(\bar{w}) \in [R]$ *iff no bad configuration is $\to_{\mathcal{P}}^A$-reachable from $\bar{w}$.*

Proof is straightforward from the definitions of U, $\mathcal{M}$, h and $[R]$.

It follows immediately that $\mathfrak{M} \models \neg \Psi_F$. To show that $\mathfrak{M} \models \Phi_{\mathcal{P}}$ we show that $\mathfrak{M} \models \varphi$ for every $\varphi \in \Phi$. For the first seven formulae in the definition of $\Phi_{\mathcal{P}}$ this involves a routine check of definitions. We show here only one case of the remaining formulae axiomatizing R.

To demonstrate $\mathfrak{M} \models (R((x * q_1) * y) \wedge P^J(x)) \rightarrow R((x * q_2) * y)$ for some $\forall_L J \ (q_1 \rightarrow q_2)$ in T assume that left-hand side of the implication is satisfied in $\mathfrak{M}$ for some assignment of the variables. That means there are $t_1, t_2 \in M$ such that $t_1 * h(q_1) * t_2 \in [R]$ and $t_1 \in [P^J]$. Furthermore, there are $\bar{w}_1, \bar{w}_2 \in Q^*$ such that $t_1 = h(\bar{w}_1)$, $t_2 = h(\bar{w}_2)$ and no bad states are $\rightarrow_{\mathcal{P}}^A$-reachable from $w_1 \ q_1 \ w_2$. Now, transition by the rule $\forall_L J \ (q_1 \rightarrow q_2)$ is possible from $\bar{w}_1 \ q_1 \ \bar{w}_2$, resulting in the configuration $\bar{w}_1 \ q_2 \ \bar{w}_2$, from which it is still the case that no bad configurations are reachable. This implies $h(\bar{w}_1) \ h(q_2) \ h(\bar{w}_2) \in [R]$, and therefore $\mathfrak{M} \models (R((x * q_1) * y) \wedge P^J(x)) \rightarrow R((x * q_2) * y)$. The remaining cases are tackled in a similar way. $\square$.

4.3 FCM is stronger than monotone abstraction

For some parameterized systems the method based on monotone abstraction may fail to establish safety even though it may actually hold. The reason for this is a possible *overapproximation* of the set of reachable states as a result of abstraction. A simple example of such a case is given in [2]. The parameterized system (Q, T) where $Q = \{q_0, q_1, q_2, q_3, q_4\}$ and where T includes the following transition rules

1. $\forall \{q_0, q_1, q_4\} : q_0 \rightarrow q_1$
2. $q_1 \rightarrow q_2$
3. $\forall_L \{q_0\} : q_2 \rightarrow q_3$
4. $q_3 \rightarrow q_0$
5. $\exists_{LR} \{q_2\} : q_3 \rightarrow q_4$
6. $q_4 \rightarrow q_3$

satisfies mutual exclusion for state q_4, but this fact can not be established by the monotone abstraction method from [1]. However, using first-order encoding presented above and the finite model finder we have verified mutual exclusion for this system, demonstrating that FCM method is stronger than monotone abstraction. Mace4 has found a finite countermodel of the size 6 in 341s. See details in [18] and the Appendix.

The issue of overapproximation has been addressed in [2] where two refinements of the monotonic abstraction method were proposed. One resulted in an exact context-sensitive symbolic algorithm which allows one to compute exact symbolic representations of predecessor configurations, but the termination of which is not guaranteed. On the other hand, an approximated context-sensitive symbolic algorithm is also proposed and while guaranteed to terminate, may still lead to overapproximation. One can show the relative completeness of the FCM method with respect to both algorithms for the case of safety verification. In both algorithms the safety is established when a finite representation of a set U of configurations backwards rechable from unsafe states, is obtained upon an algorithm termination. In both cases such a set U can be shown is regular, and therefore one can apply the arguments used in the proof of Theorem 1. We postpone the detailed presentation till another occassion, but would like to emphasize that the main reason for the relative completeness here is a mere *existence* of the regular sets of configurations subsuming all reachable configurations and disjoint with unsafe configurations.

5 Regular model checking

The result of the previous section may appear rather narrow and related to a specific class of parameterized systems. The verification of safety for this class can be re-formulated for, and dealt with the traditional *regular model checking* approach[21]. In this section we extend our relative completeness result and show that whenever safety for a parameterized system can be established by the regular model checking approach then it can also be verified by the finite countermodel finding method.

We start with the basics of the traditional regular model checking approach, borrowing standard definitions largely from [14]. A finite automaton is a tuple $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, where Q is a finite set of states, Σ is a finite alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is a set of transitions, $q_0 \in Q$ is an initial state and $F \subseteq Q$ is a set of final (accepting) states. M is deterministic automaton if $\forall q \in Q \ \forall a \in \Sigma$ there esists at most one q' such that $\langle q, a, q' \rangle \in \delta$. With every finite automaton we associate a transition relation $\rightarrow \ \subseteq \ Q \times \Sigma^* \times Q$ which is defined as the smallest relation satisfying: (1): $\forall q \in Q; q \rightarrow^\epsilon q$, (2) if $\langle q, a, q' \rangle \in \delta$, then $q \rightarrow^a q'$, (3) if $q \rightarrow^w q'$ and $q' \rightarrow^a q''$ then $q \rightarrow^{wa} q''$. The language recognized by the automaton M is defined as $L(M) = \{w \mid \exists q' \in F \ \wedge \ \wedge q_0 \rightarrow^w q'\}$.

Let Σ be a finite alphabet and $\epsilon \notin \Sigma$. Let $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$. A *finite transducer* over Σ is a tuple $\tau = \langle Q, \Sigma_\epsilon^* \times \Sigma_\epsilon^*, \delta, q_0, F \rangle$, where Q is a finite set of states, $\delta \subseteq Q \times \Sigma_\epsilon \times \Sigma_\epsilon \times Q$ a set of transitions, $q_0 \in Q$ is an initial state, and $F \subseteq Q$ is a set of final (accepting) states. The transition relation $\rightarrow \subseteq Q \times \Sigma^* \times \Sigma^* \times Q$ is defined as the smallest relation staisfying: (1) $q \rightarrow^{\epsilon, \epsilon} q$ for every $q \in Q$, (2) if $\langle q, a, b, q' \rangle \in \delta$, then $q \rightarrow^{a,b} q'$, and (3) if $q \rightarrow^{w,u} q'$ and $q' \rightarrow^{a,b} q''$, then $q \rightarrow^{wa,ub} q''$. With every transducer τ we associate a binary relation $r_\tau = \{\langle w, u \rangle \mid \exists q' \in F \ \wedge \ q_0 \rightarrow^{w,u} q'\}$. Let r_τ^* denote the reflexive and transitive closure of r_τ.

The verification of safety properties in the framework of regular model checking proceeds as follows. The set of initial states of the parameterized (or infinite state) system is presented by an effectively given (by a finite automaton) *regular* language $Init$. The set of "bad", or unsafe states is described by another regular language Bad. One-step transitions of the system to be verified are presented by a transducer relation r_τ (for some finite state transducer τ). The verification of safety property ("never get into the bad states") is reduced to the following

Problem 1. Given regular sets $Init$ and Bad and a finite transducer τ, does $r_\tau^*(Init) \cap Bad = \emptyset$ hold?

Regular model checking (RMC) is one of the most general methods for formal verification of parameterized and infinite state systems [21,4]. One of the issues with the method is that the termination of the computation of transitive closure $r_\tau^*(Init)$ is not guaranteed. To alleviate this issue, various acceleration methods have been proposed. We show that the finite countermodel finding method is actually as powerful as any variant of RMC, the only assumption to guarantee its termination is the *existence* of a regular set R subsuming $r_\tau^*(Init)$ and being disjoint with Bad.

5.1 From regular model checking to first-order disproving

In this subsection we show how to reduce the generic regular model checking question posed in the Problem 1 above to the problem of disproving of a formula from classical first-order predicate logic. Solution of the latter problem is then delegated to the generic automated finite model finding procedure.

Assume we are given

- a finite state automaton $M_1 = \langle Q_1, \Sigma, \delta_1, q_{0_1}, F_1 \rangle$ recognizing a regular language $Init$;
- a finite state automaton $M_2 = \langle Q_2, \Sigma, \delta_2, q_{0_2}, F_2 \rangle$ recognizing a regular language Bad;
- a finite state length-preserving transducer $\tau = \langle Q, \Sigma^* \times \Sigma^*, \delta, q_0, F \rangle$ representing the transition relation r_τ;

Assume also (without loss of generality) that sets Q_1, Q_2, Q, Σ are disjoint.
Now define a set of formulae of first-order predicate logic as follows. In fact, it is a formalization of the above definition of $\rightarrow$ within first-order predicate logic.
  The vocabulary consists of

- constants for all elements of $\Sigma \cup Q_1 \cup Q_2 \cup Q$ plus one distinct constant e;
- a binary functional symbol $*$;
- unary relational symbols R, $Init$ and Bad;
- a binary relational symbol $Trans$;
- a ternary relational symbol $T^{(3)}$;
- a 4-ary relational symbol $T^{(4)}$;

Let Φ be the set of the following formulae, which are all assumed to be universally closed:

1. $(x * y) * z = x * (y * z)$
2. $T^{(3)}(q, e, q)$ for all $q \in Q_1 \cup Q_2$;
3. $T^{(3)}(q, a, q')$ for all $(q, a, q') \in \delta_1 \cup \delta_2$;
4. $T^{(3)}(x, y, z) \wedge T^{(3)}(z, v, w) \rightarrow T^{(3)}(x, y * v, w)$
5. $\vee_{q \in F_1} T^{(3)}(q_{0_1}, x, q) \rightarrow Init(x)$
6. $\vee_{q \in F_2} T^{(3)}(q_{0_2}, x, q) \rightarrow Bad(x)$
7. $T^{(4)}(x, e, e, x)$
8. $T^{(4)}(q, a, b, q')$ for all $(q, a, b, q') \in \delta$
9. $T^{(4)}(x, y, z, v) \wedge T^{(4)}(v, y', z', w) \rightarrow T^{(4)}(x, y * y', z * z', w)$
10. $Trans(x, y) \leftrightarrow \vee_{q \in F} T^{(4)}(q_0, x, y, q)$
11. $Init(x) \rightarrow R(x)$
12. $R(x) \wedge Trans(x, y) \rightarrow R(y)$

Proposition 3 (adequacy of Init and Bad translations).

  If $w \in Init$ then $\Phi \vdash Init(t_w)$
  If $w \in Bad$ then $\Phi \vdash Bad(t_w)$

Proof For $w = s_1, \ldots s_n \in Init$ we have w is accepted by the finite automaton M_1, which means there is a sequence of states $q_{0_1}, q_1, \ldots q_n$ with $q_n \in F_1$ such that $\langle q_i, s_i, q_{i+1} \rangle \in \delta_1$ for $i = 0, \ldots n - 1$. By the definition of Φ (clause 3) all formulae $T(q_i, s_i, q_{i+1})$ are in Φ. Together with clause 4, this gives $\Phi \vdash T(q_{0_1}, t_w, q_n)$. This with $q_n \in F_1$ and using clause 5 entails $\Phi \vdash Init(t_w)$. The second statement is proved in the same way. $\square$

Proposition 4 (adequacy of encoding). *If $w \in r_\tau^*(Init)$ then $\Phi \vdash R(t_w)$*

Proof. Easy induction on the length of transition sequences.

- *Induction Base Case.* Let $w \in Init$. Then $\Phi \vdash Init(t_w)$ (by Proposition 3), and, further, $\Phi \vdash R(t_w)$ (using clause 11).
- *Induction Step Case.* Let $w \in r_\tau^{n+1}(Init)$. Then there exists w' such that $w' \in r_\tau^n(Init)$ and $\langle w', w \rangle \in r_\tau$. By the induction assumption $\Phi \vdash R(t_{w'})$. Further, by an argument analogous to the proof in Proposition 3, $\langle w', w \rangle \in r_\tau$ entails $\Phi \vdash T(q_0, t_{w'}, t_w, q)$ for some $q \in F$. It follows, using clause 10, that $\Phi \vdash Trans(t_{w'}, t_w)$. From this, the clause 12 and the induction assumption $\Phi \vdash R(t_{w'})$ follows.

Corollary 2. *If $r_\tau^*(Init) \cap Bad \neq \emptyset$ then $\Phi \vdash \exists x(R(x) \wedge Bad(x))$.*

The Corollary 2 serves as a formal underpinning of the proposed verification method. In order to prove safety, that is $r_\tau^*(Init) \cap Bad = \emptyset$ it suffices to demonstrate $\Phi \nvdash \exists x(R(x) \wedge Bad(x))$, or equivalently, to *disprove* $\Phi \rightarrow \exists x(R(x) \wedge Bad(x))$. We delegate this task to the finite model finding procedures, which search for the *finite* countermodels for $\Phi \rightarrow \exists x(R(x) \wedge Bad(x))$.

5.2 Relative completeness with respect to RMC

As highlighted earlier, searching for *finite* countermodels to disprove non-valid first-order formulae may not always lead to success, because for some formulae countermodels are inevitably *infinite*. In this subsection we show that it is not the case for the first-order encodings of the problems which can be positively answered by RMC, and therefore such problems can also be resolved positively by the proposed finite countermodel finding method, provided a complete finite model finding procedure is used.

   Assume that RMC answers positively the question of Problem 1 above. In the RMC approach the positive answer follows from producing a *regular* set $\mathcal{R}$ such that $r_\tau^*(Init) \subseteq \mathcal{R}$ and $\mathcal{R} \cap Bad = \emptyset$. We show that in such a case there always exists a *finite* countermodel for $\Phi \rightarrow \exists x(R(x) \wedge Bad(x))$.

Theorem 2 (relative completeness). *Let $Init$ and Bad be regular sets given by recognizing finite automata M_1 and M_2, and τ be a finite state transducer. Let Φ be a first-order formula defined above. If there exists a* regular *set $\mathcal{R}$ such that $r_\tau^*(Init) \subseteq \mathcal{R}$ and $\mathcal{R} \cap Bad = \emptyset$ then there exists a finite countermodel for $\Phi \rightarrow \exists x(R(x) \wedge Bad(x))$*

Proof

Since $\mathcal{R}$ is regular, according to the algebraic characterization of regular sets, there exists a finite monoid $\mathcal{M} = (M, \circ)$, a subset $S \subseteq M$ and a homomorphism $h : \Sigma^* \to \mathcal{M}$ such that $\mathcal{R} = \{\bar{w} \mid \bar{w} \in \Sigma^* \wedge h(\bar{w}) \in S\}$.

We take $M \cup Q_1 \cup Q_2$ to be domain of the required finite model, and then define interpretations as follows.

- For $a \in \Sigma$ $[a] = h(a)$;
- $[e] = \underline{1}$, where $\underline{1}$ is an unit element of the monoid;
- $[*]$ is a monoid operation $\circ$;
- Interpretations of T^3 and T^4 are defined inductively, as the least subsets of tuples satisfying, respectively, formulae (2)-(4) and (7) - (9)(and assuming all interpretations given above);
- Interpretations of $Init$ and Bad are defined to be the least subsets satisfying (5) and (6), respectively (assuming all interpretations above);
- Interpretation of Trans is defined by (10)(assuming all interpretations above);
- Interpretation of R is S.

Now it is straightforward to check that such defined a *finite* model indeed satisfies $\Phi \wedge \neg \exists x (R(x) \wedge Bad(x))$. Checking that Φ is satisfied is by routine inspection of the definitions. To check that $\neg \exists x (R(x) \wedge Bad(x))$ is satisfied, assume the opposite holds. So there exists an element a of the monoid $\mathcal{M}$ such that $a \in [R]$ and $a \in [Bad]$. Then, for a word $w \in \Sigma^*$ such that $h(w) = a$, we have $w \in \mathcal{R} \cap Bad \neq \emptyset$, which contradicts with the assumption of the theorem. $\square$.

5.3 Optimizations

In many cases (i.e. in many subclasses of verification tasks), the transition relation and/or the sets of 'initial' and 'bad' states are described not by finite state transducers/automata, but in more explicit and simpler ways, e.g. by rewriting rules for transitions and simple grammars generating sets of states. In such a cases, first-order translations can be made simpler and the whole procedure more efficient. Our treatment of the case of parameterized linear automata in Section 4 can be seen as an illustration of such a modification.

5.4 Experimental results

In the experiments we used the finite model finder Mace4[20] within the package Prover9-Mace4, Version 0.5, December 2007. It is not the latest available version, but it provides with convenient GUI for both the theorem prover and the finite model finder. The system configuration used in the experiments: Microsoft Windows XP Professional, Version 2002, Intel(R) Core(TM)2 Duo CPU, T7100 @ 1.8Ghz 1.79Ghz, 1.00 GB of RAM. The time measurements are done by Mace4 itself, upon completion

of the model search it communicates the CPU time used. The table below lists the parameterised/infinite state protocols together with the references and shows the time it took Mace4 to find a countermodel and verify a safety property. The time shown is an average of 10 attempts.

Protocol	Reference	Time
Token passing (non-optimized)	[14]	0.12s
Token passing (optimized)	[14]	0.01s
Mutual exclusion I	[1] and 3	0.03s
Mutual exclusion II	[2] and 4.2	341s
Bakery	[21]	0.03s
Paterson$^-$	[8] and 5.5	0.77s

5.5 Beyond regular model checking

The method of verification via disproving (countermodel finding) can be applied also to classes of problems where traditional regular model checking is not applicable. Consider, for example, the case where the set of initial states is not regular, so the standard algorithms of RMC are not applicable. In the paper [8] an extension of regular model checking is proposed, which is capable to tackle some non-regular cases. Not claiming any kind of completeness (yet!) we show in this subsection that a case study example from [8] can be (partially, as for now) tackled by the finite countermodel finding method too. Consider the following string rewriting system over alphabet $\{0, 1\}$, which is an encoding of the parameterized Paterson mutual exclusion algorithm from [8]:

1. $x01y \to x10y$ where $x \in 0^*$, $y \in (1 + 0)^*$
2. $x101y \to x110y$ where $x \in (1 + 0)^*$, $y \in 1^*$
3. $x001y \to x010y$ where $x, y \in (1 + 0)^*$
4. $x0 \to 0x$ where $x \in (1 + 0)^*$

The safety condition for this rewriting system is 'Starting from any string of the form $0^n 1^n$ no string from the set $(0 + 1)^* 00$ is reachable' (mutual exclusion of the original Paterson algorithm). In [8] it is shown that the extension of RMC proposed there can successfully verify the condition.

Following the translation from the Section 4 we encode the string rewriting system into a first-order formula Φ. Since the set of initial states is not regular, the formula contains a part specifying the generation of initial states by a context-free grammar: $R(e) \wedge R(x) \to R((0 * x) * 1)$.

In the experiments we failed to verify the correctness condition for the Paterson algorithm, however, for the reduced string rewriting system Paterson$^-$, containing only the rules 1,2,4 we have verified safety condition above. Mace4 has found a finite countermodel of size 8 in $0.77s$. The details can be found in [18].

6 Related work

As mentioned Section 1 the approach to verification using the modeling of protocol executions by first-order derivations and together with countermodel finding for disproving was introduced within the research on the formal analysis of cryptographic protocols. It can be traced back to the early papers by Weidenbach [23] and by Selinger [22]. In [23] a decidable fragment of Horn clause logic has been identified for which resolution-based decision procedure has been proposed (disproving by the procedure amounts to the termination of saturation without producing a proof). It was also shown that the fragment is expressive enough to encode cryptographic protocols and the approach has been illustrated by the automated verification of some protocols using the SPASS theorem prover. In [22], apparently for the first time, explicit building of finite countermodels has been proposed as a tool to establish correctness of cryptographic protocols. It has been illustrated by an example, where a countermodel was produced manually, and the automation of the process has not been disscussed. The later work by Goubault-Larrecq [11] has shown how a countermodel produced during the verification of cryptographic protocols can be converted into a formal induction proof. Also, in [11] different approaches to model building have been discussed and it was argued that an implicit model building procedure using alternating tree automata is more efficient in the situations when no small countermodels exist. Very recently, in the paper [15] by J. Jurgens and T. Weber, an extension of Horn clause logic was proposed and the soundness of a countermodel finding procedure for this fragement has been shown, again in the context of cryptographic protocol verification. Furthermore, in [15] an approach to the verification of parameterized cryptoprotocols is proposed.

The work we reported in this paper differs from all the approaches mentioned previously in two important aspects. Firstly, to the best of our knowledge, none of the previous work addressed verification via countermodel finding applied outside of the area of cryptographic protocols (that includes the most recent work [13] we are aware of). Secondly, the (relative) completeness for the classes of verification tasks has not been addressed in previous work.

The encoding of infinite state systems in first-order predicate logic is used in the MCMT deductive symbolic model checker [9,10]. While principles of encoding used in MCMT are very much similar to these we consider in the present paper, the verification procedure is quite different. The core algorithm of MCMT relies on a symbolic backwards reachability procedure, in which first-order formulae are used for the symbolic representation of the sets of configuration. During the execution the reachability procedure may call the external logic engine (SMT solver) multiple times, up to several hundreds for some examples as reported in [9]. In the FCM method we presented here the verification procedure is much simpler and is just a reduction (or compilation) to a *single* problem in logic, which then is resolved via single call to the external logic engine (finite model builder).

In a more general context, the work we present in this paper is related to the concepts of *proof by consistency* [16], and *inductionless induction* [6] and can be seen as an investigation into the power of these concepts in the particular setting of the verification of parameterized systems via finite countermodel finding.

7 Conclusion

We have shown how to apply generic finite model finders in the parameterized verification of linear arrays of finite automata models, have demonstrated the relative completeness of the method, and have illustrated its practical efficiency. Further, we have shown that the verification via finite countermodel finding is at least as powerful as the standard regular model checking for the verification of safety properties. Inspection of the proofs of relative completeness reveals that the key reason for the completeness is the existence of regular sets separating the reachable and bad (unsafe) states. We conclude with the very general claim that, for any parameterized system, for which there exists a regular set separating reachable and unsafe states, its correctness can be demonstrated by a finite countermodel finding method. Formal instantiations of this claim for particular classes of problems remains a subject of ongoing and future work. In particular, the extension of the results presented in this paper to the case of *tree* regular model checking looks quite straightforward. More speculative and intriguing is a possibility to use *infinite* model building procedures [5] for parameterized verification. Further investigation of practical efficiency and scalability of the method is also an important direction for future work.

8 Acknowledgments

References

1. Parosh Aziz Abdulla, Giorgio Delzanno, Noomene Ben Henda, Ahmed Rezine. Monotonic Abstraction: on Efficient Verification of Parameterized Systems. *Int. J. Found. Comput. Sci.* 20(5): 779-801 (2009)
2. Parosh Aziz Abdulla, Girogio Delzanno, Ahmed Rezine. Approximated Context-Sensitive Analysis for Parameterized Verification. Lecture Notes in Computer Science, 2009, Volume 5522, 41-56
3. Parosh Aziz Abdulla, Jonsson B. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91-101, June 15, 1996.
4. Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Mayank Saksena, A Survey of Regular Model Checking, In Proc. of CONCUR'04, volume 3170 of LNCS, pp 35–58, 2004.
5. R. Caferra, A. Leitsch, N. Peltier, *Automated Model Building*, Applied Logic Series, 31, Kluwer, 2004.
6. H. Comon. Inductionless induction. In R. David, ed. *2nd Int. Conf. in Logic for Computer Science: Automated Deduction. Lecture Notes*, Chambery, Uni de Savoie, 1994.
7. G. Delzanno. Constraint-based Verification of Parametrized Cache Coherence Protocols. *Formal Methods in System Design*, 23(3):257–301, 2003.
8. Dana Fisman and Amir Pnueli, Beyond Regular Model Checking, In Proc. of FSTTCS01, volume 2245 of LNCS, 2001.
9. S.Ghilardi and S.Ranise. MCMT: A Model Checker Modulo Theories. Lecture Notes in Computer Science, 2010, Volume 6173/2010, 22–29.

10. S.Ghilardi, E.Nikolini, S.Ranise and D.Zucchelli. Towards SMT Model-Checking of Array-based Systems. In IJCAR, LNCS, 2008

11. J. Goubault-Larrecq. Towards producing formally checkable security proofs, automatically. In: Computer Security Foundations (CSF), pp. 224-238 (2008)

12. J. Goubault-Larrecq. "Logic Wins!". In ASIAN'09, LNCS 5913, pages 1-16. Springer, 2009.

13. Joshua Guttman, Security Theorem via Model Theory, arXiv:0911.2036, November 2009.

14. Peter Habermehl, Tomas Vojnar, Regular Model Checking Using Inference of Regular Languages, Electronic Notes in Theoretical Computer Science (ENTCS) , Volume 138 , Issue 3 (December 2005) , pp 21–36, 2005

15. J. Jurjens and T. Weber, Finite Models in FOL-Based Crypto-Protocol Verification, P. Degano and L. Vigan'o (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 155-172, 2009.

16. D. Kapur and D.R. Musser. Proof by consistency. *Artificial Intelligence*, 31:125–157, 1987.

17. A. Lisitsa Reachability as deducibility, finite countermodels and verification. In preProceedings of AVOCS 2009, Technical Report of Computer Science, Swansea University, CSR-2-2009, pp 241-243.

18. A. Lisitsa Verfication via countermodel finding
`http://www.csc.liv.ac.uk/~alexei/countermodel/`

19. A. Lisitsa Reachability as deducibility, finite countermodels and verification. A conference version of [17], 14pp, (to appear in Proc. ATVA2010)

20. W. McCune Prover9 and Mace4 `http://www.cs.unm.edu/~mccune/mace4/`

21. M. Nilsson. Regular Model Checking. Acta Universitatis Upsaliensis. Uppsala Dissertations from the Faculty of Science and Technology 60. 149 pp. Uppsala. ISBN 91-554-6137-9, 2005.

22. P. Selinger, Models for an adversary-centric protocol logic. Electr. Notes Theor. Comput. Sci. 55(1) (2001)

23. C. Weidenbach, Towards an Automatic Analysis of Security Protocols in First-Order Logic, in H. Ganzinger (Ed.): CADE-16, LNAI 1632, pp. 314–328, 1999.